Treat the Entire Inherited Process Context as Unvalidated Input

William L. Fithen, Software Engineering Institute [vita[3]]

2005-10-03 L4 / D/P[4]

Inherited process context that is not validated like other inputs can introduce vulnerability.

Description

It is necessarily the case that when a program starts execution, it inherits an execution context that is potentially malicious. The only parts of the execution context that can be malicious are those aspects that are both dynamic and subject to influence by an adversary. That is, anything that an invoking adversary could change offers the potential of being a trap for the program. Static aspects of the execution environment are accommodated in the normal engineering approaches.

The only approach to this situation is to design a protocol to establish trust in the context. This includes

- correcting or discarding aspects of the context
- validating those aspects of the context that cannot be corrected

Note that the adversary can change some aspects of the execution context while the program is running. That is not covered here. Those are generally considered time and state problems.

Dynamic Assembly

Many operating systems and programming languages support assembly of executable programs immediately before or during execution. The pieces from which the ultimate executable is composed must be acquired from many places. The most common are

- files from local filesystems
- files from remote (network) filesystems
- files from web, code, or application servers on the network

The specific locations for these component files are generally configured into the operating systems or runtime environments of the programming languages. In some cases, these locations are incorrectly configured or can be fooled into loading pieces from vulnerable locations—locations where adversaries can pre-position maliticous code.

Environments where this has been known to occur include the following:

- Virtually all modern UNIX operating systems have support for shared object libraries. An adversary may be able to fool the system into loading a shared library before or instead of those the engineer of the program expects.
- All modern versions of Microsoft Windows operating systems have support for dynamically loaded libraries. As in the case of UNIX, adversaries may be able to load their DLLs before or instead of those expected.
- Many modular scripting languages, including
  - PERL,
  - Ruby, and
  - Python.

[3] http://buildsecurityin.us-cert.gov/bsi/about_us/authors/320-BSI.html (Fithen, William L.)

- Java and .NET include dynamic assembly is an integral part of the platforms themselves. That is, while most of the execution environments described above use dynamic assembly for efficiency, Java and .NET use it functionally. Consequently, it cannot be avoided.

References

[CA-1995-14] cert.org. *CERT® Advisory CA-1995-14 Telnetd Environment Vulnerability*. http://www.cert.org/advisories/CA-1995-14.html (1997).

[Gosling 05] Gosling, James; Joy, Bill; Steele, Guy; & Bracha, Gilad. *The Java Language Specification, Third Edition*. Boston, MA: Addison-Wesley Professional, 2005. http://java.sun.com/docs/books/jls/.

[Grubb 02] Grubb, Steven. *A Survey of Process Environments*. http://www.web-insights.net/env_audit/environments.pdf (2002).

[McClure 99] McClure, Stuart; Scambray, Joel; & Kurtz, George. *Hacking Exposed: Network Security Secrets & Solutions*, 316-317. Computing Mcgraw-Hill, 1999.

[Thompson 05] Thompson, Herbert & Chase, Scott. *The Software Vulnerability Guide*, 211-222. Charles River Media, 2005.

[VU#602625] cert.org. *Vulnerability Note VU#602625: KTH Kerberos environment variables krb4proxy and KRBCONFDIR may be used insecurely*. http://www.kb.cert.org/vuls/id/602625 (2001).

[VU#943633] cert.org. *Vulnerability Note VU#943633: FreeBSD can be compromised locally via signal handlers*. http://www.kb.cert.org/vuls/id/943633 (2002).

Carnegie Mellon Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

NO WARRANTY

THIS MATERIAL OF CARNEGIE MELLON UNIVERSITY AND ITS SOFTWARE ENGINEERING INSTITUTE IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[1] mailto:permission@sei.cmu.edu